
DemoSystem2004

RTES Collaboration (NSF ITR grant ACI-0121658)



**Real-Time and Embedded Technology
& Applications Symposium (IEEE)**

FALSE II Workshop

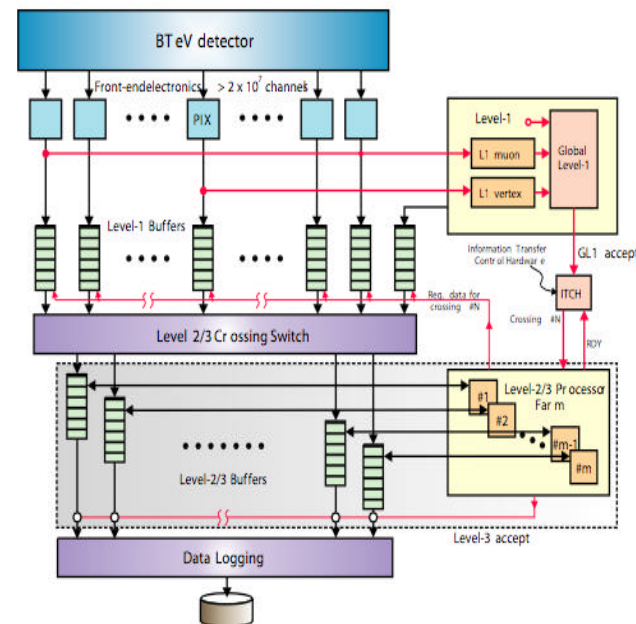
San Francisco, March 7-10, 2005

Introduction

- Our context: BTeV
 - A 20 TeraHz Real-Time System
- Our Solution
 - MIC, ARMORs, VLAs
- The demonstration system
- Demonstration
- Comments

BTeV High Energy Physics Experiment: A 20 TeraHz Real-Time System

- Input: 800 GB/s (2.5 MHz)
- Level 1
 - Lvl1 processing: 190μs
 - rate of 396 ns
 - 528 “8 GHz” G5 CPUs
 - (factor of 50 event reduction)
 - high performance interconnects
- Level 2/3:
 - Lvl 2 processing: 5 ms
 - (factor of 10 event reduction)
 - Lvl 3 processing: 135 ms
 - (factor of 2 event reduction)
 - 1536 “12 GHz” CPUs commodity networking
- Output: 200 MB/s (4 kHz) = 1-2 Petabytes/year



The Problem

- Monitoring, Fault Tolerance and Fault Mitigation are crucial
 - In a cluster of this size, processes and daemons are constantly hanging/failing without warning or notice
- Software reliability depends on
 - Physics detector-machine performance
 - Program testing procedures, implementation, and design quality
 - Behavior of the electronics (front-end and within the trigger)
- Hardware failures will occur!
 - one to a few per week






The Problem (continued)

- Given the very complex nature of this system where thousands of events are simultaneously and asynchronously cooking, issues of data integrity, robustness, and monitoring are critically important and have the capacity to cripple a design if not dealt with at the outset...

BTeV [needs to] supply the necessary level of “self-awareness” in the trigger system.

[June 2000 Project Review]

BTeV's Response: RTES

- The Real Time Embedded System Group
 - A collaboration of five institutions,
 -  University of Illinois
 -  University of Pittsburgh
 -  University of Syracuse
 -  Vanderbilt University (PI)
 -  Fermilab
 - NSF ITR grant ACI-0121658
 - Physicists and Computer Scientists/Electrical Engineers at BTeV institutions with expertise in
 - High performance, real-time system software and hardware,
 - Reliability and fault tolerance,
 - System specification, generation, and modeling tools.
 - Working on fault management in large computing clusters
-

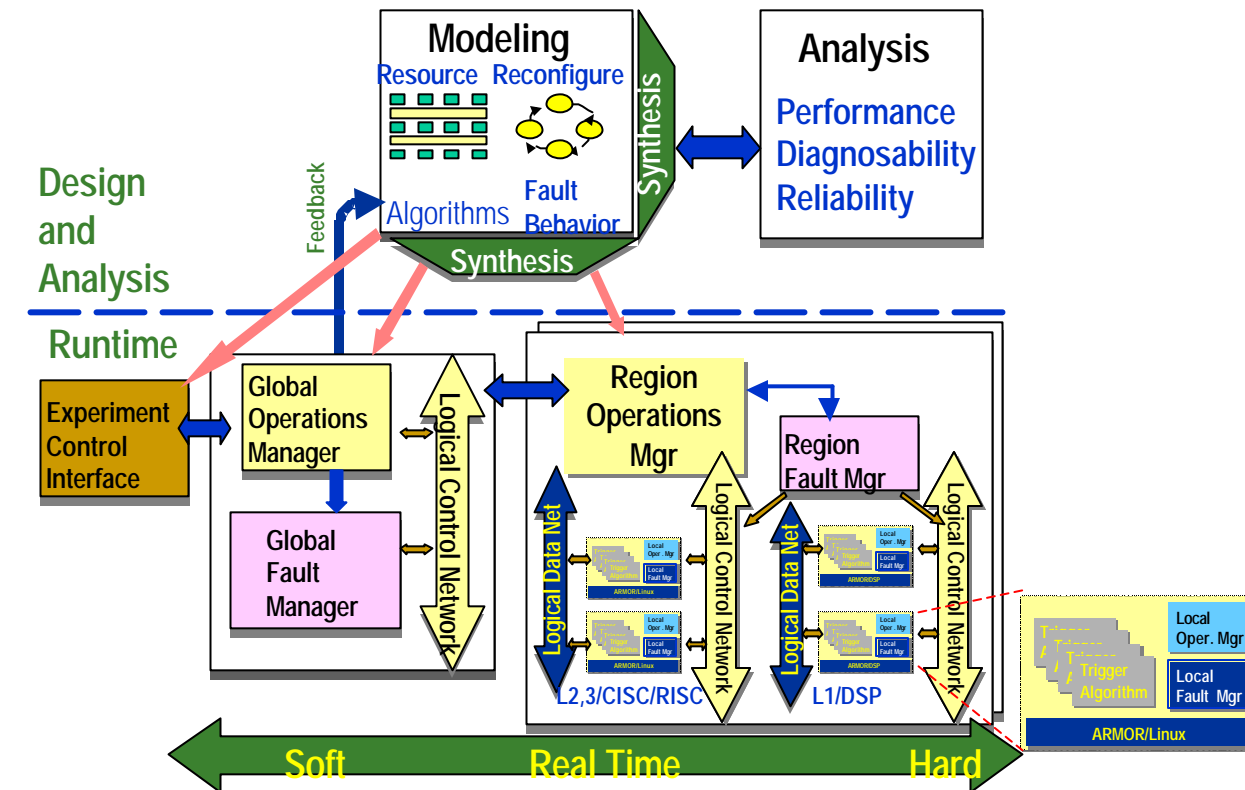
RTES Goals for BTeV

- High availability
 - Fault handling infrastructure capable of
 - Accurately identifying problems (where, what, and why)
 - Compensating for problems (shift the load, changing thresholds)
 - Automated recovery procedures (restart / reconfiguration)
 - Accurate accounting
 - Extensibility (capturing new detection/recovery procedures)
 - Policy driven monitoring and control
 - Dynamic reconfiguration
 - adjust to potentially changing resources
-

RTES Goals for BTeV (continued)

- Faults must be detected/corrected ASAP
 - semi-autonomously
 - with as little human intervention as possible
 - distributed and hierarchical monitoring and control
- Life-cycle maintainability and evolvability
 - to deal with new algorithms, new hardware and new versions of the OS

The RTES Solution



RTES Deliverables

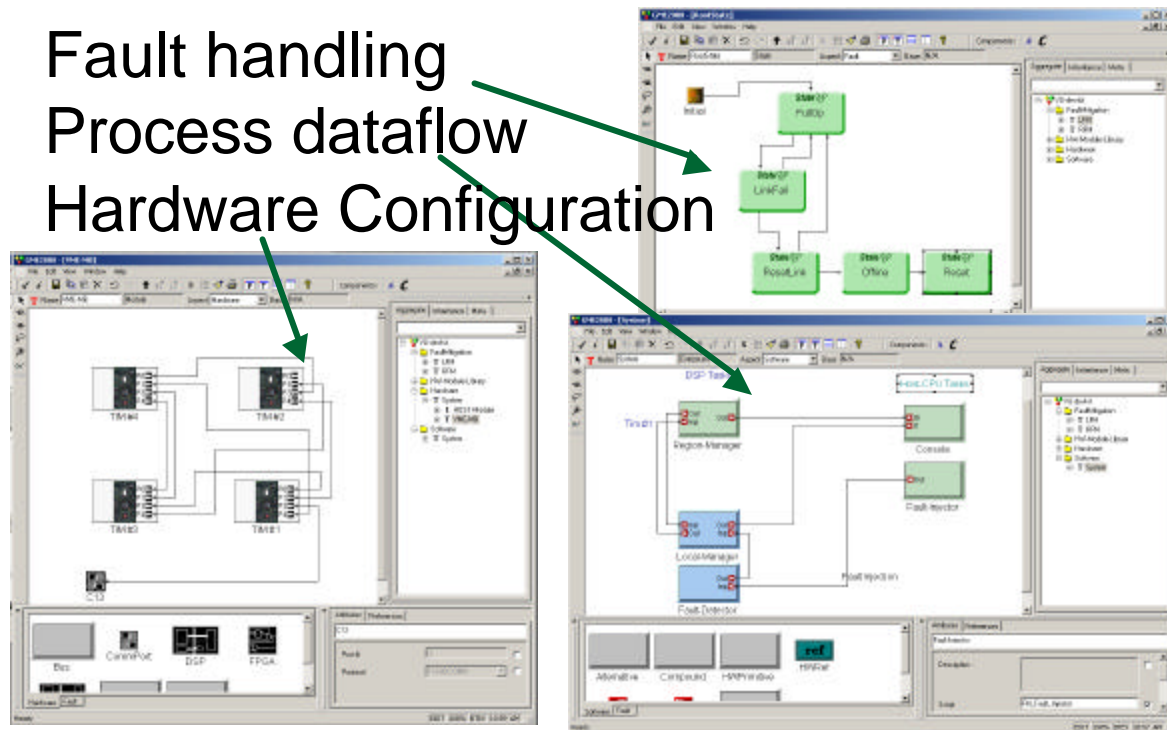
- A hierarchical fault management system and toolkit:
 - Model Integrated Computing
 - GME (Generic Modeling Environment) system modeling tools
 - and application specific “graphic languages” for modeling system configuration, messaging, fault behaviors, user interface, etc.
 - ARMORs (Adaptive, Reconfigurable, and Mobile Objects for Reliability)
 - Robust framework for detection and reaction to faults in processes
 - VLAs (Very Lightweight Agents for limited resource environments)
 - To monitor/mitigate at every level
 - DSP, Supervisory nodes, Linux farm, etc.

Configuration through Modeling

- Multi-aspect tool, separate views of
 - Hardware – components and physical connectivity
 - Executables – configuration and logical connectivity
 - Fault handling behavior using hierarchical state machines
 - Model interpreters can generate the system image
 - At the code fragment level (for fault handling)
 - Download scripts and configuration
 - Modeling “languages” are application specific
 - Shapes, properties, associations, constraints
 - Appropriate to application/context
 - System model
 - Messaging
 - Fault mitigation
 - GUI, etc.
-

Modeling Environment

Fault handling
Process dataflow
Hardware Configuration



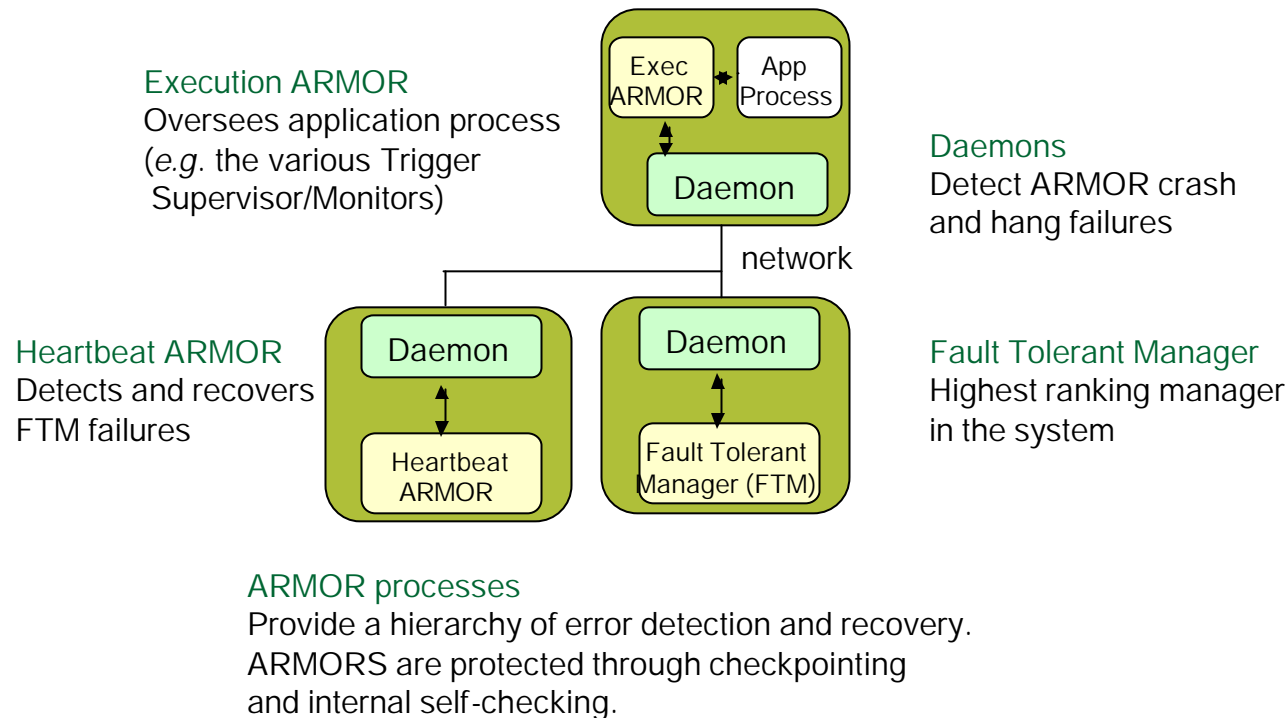
ARMOR

- **Adaptive Reconfigurable Mobile Objects of Reliability:**
 - Multithreaded processes composed of replaceable building blocks
 - Provide error detection and recovery services to user applications
 - **Hierarchy of ARMOR processes form runtime environment:**
 - System management, error detection, and error recovery services distributed across ARMOR processes.
 - ARMOR Runtime environment is itself self checking.
 - **3-tiered ARMOR support of user application**
 - Completely transparent and external support
 - Enhancement of standard libraries
 - Instrumentation with ARMOR API
-

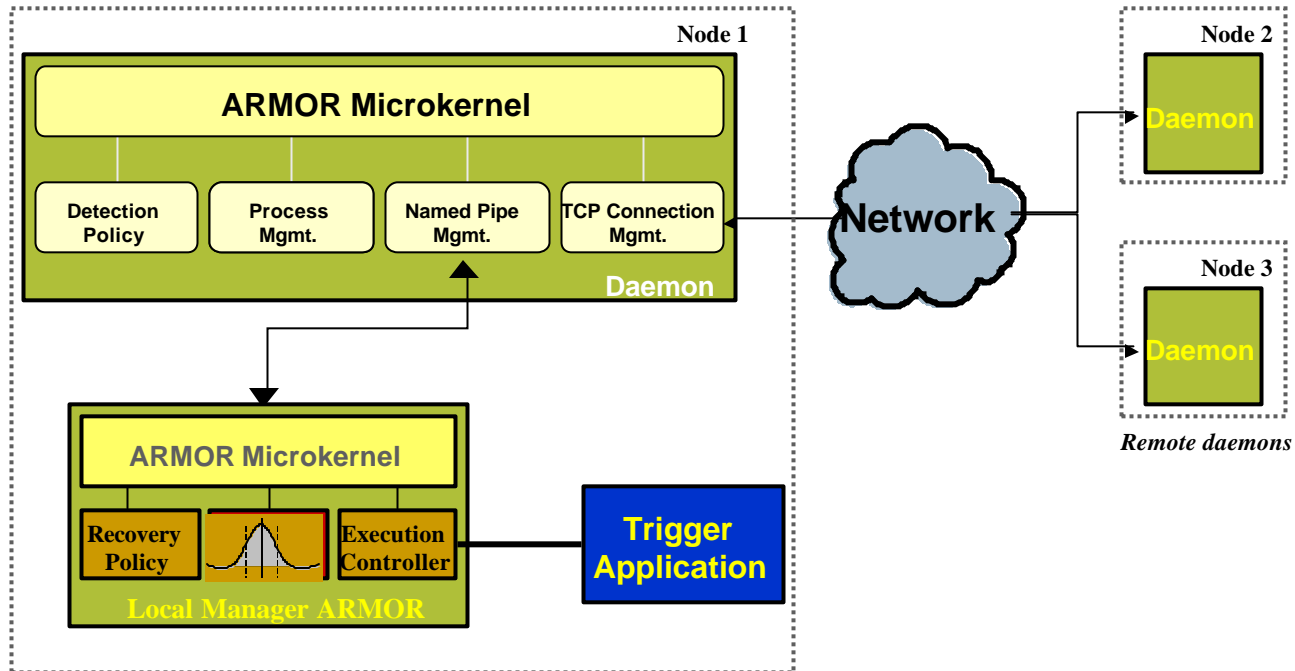
ARMOR Scalable Design

- ARMOR processes designed to be *reconfigurable*:
 - Internal architecture structured around event-driven modules called *elements*.
 - Elements provide functionality of the runtime environment, error-detection capabilities, and recovery policies.
 - Deployed ARMOR processes contain only elements necessary for required error detection and recovery services.
 - ARMOR processes *resilient* to errors by leveraging multiple detection and recovery mechanisms:
 - Internal self-checking mechanisms to prevent failures from occurring and to limit error propagation.
 - State protected through checkpointing.
 - Detection and recovery of errors.
 - ARMOR runtime environment *fault-tolerant* and *scalable*:
 - 1-node, 2-node, and *N*-node configurations.
-

ARMOR System: Basic Configuration



ARMOR: Internal Structure

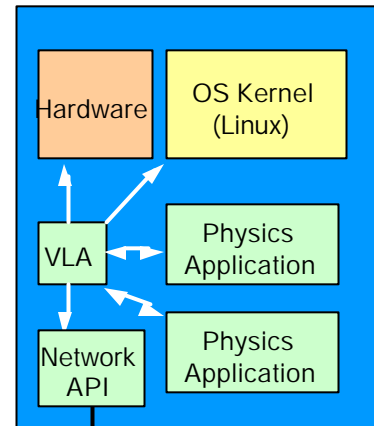


Adaptive, **R**econfigurable, and **M**obile **O**bjects for **R**eliability

Very Lightweight Agents

- Minimal footprint
- Platform independence
 - Employable everywhere in the system!
- Monitors hardware and software
- Handles fault detection & communications with higher level entities

Level 2/3 Farm Nodes
(Linux)



L2/L3 Manager Nodes
(Linux)

BTeV Prototype Farms at Fermilab

L2/3 Trigger Workers



L2/3 Trigger Racks F1-F5

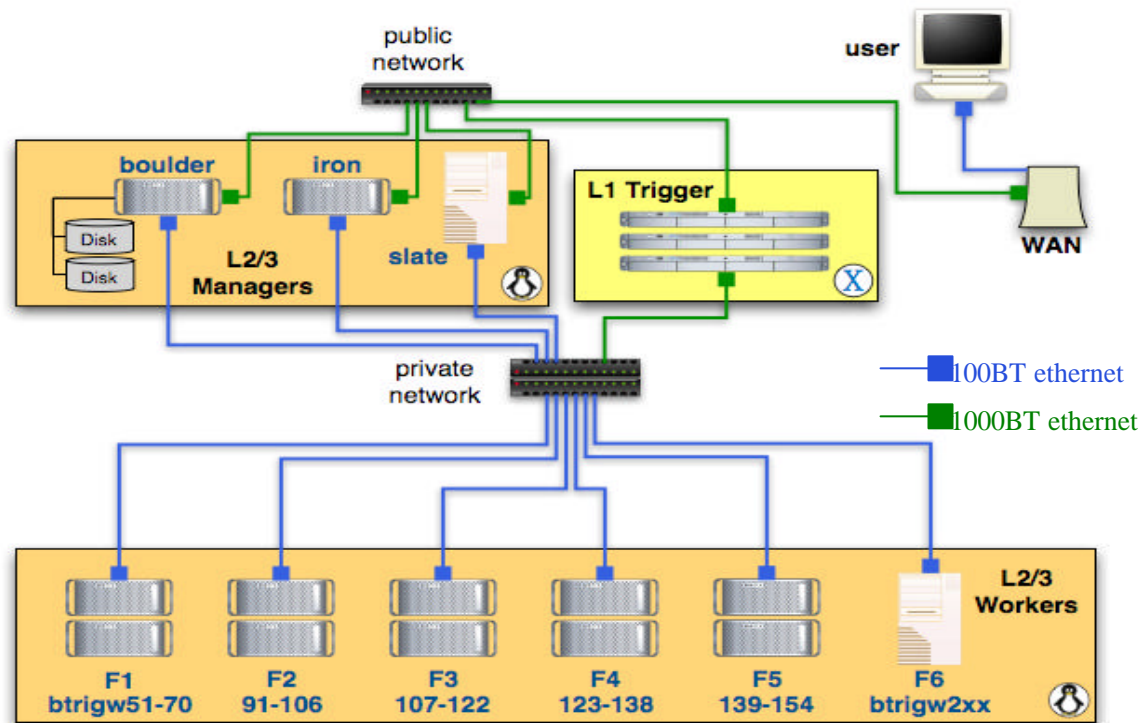
L1 Trigger Farm

Infiniband switch



16-node Apple G5 farm

L2/3 Prototype Farm Setup



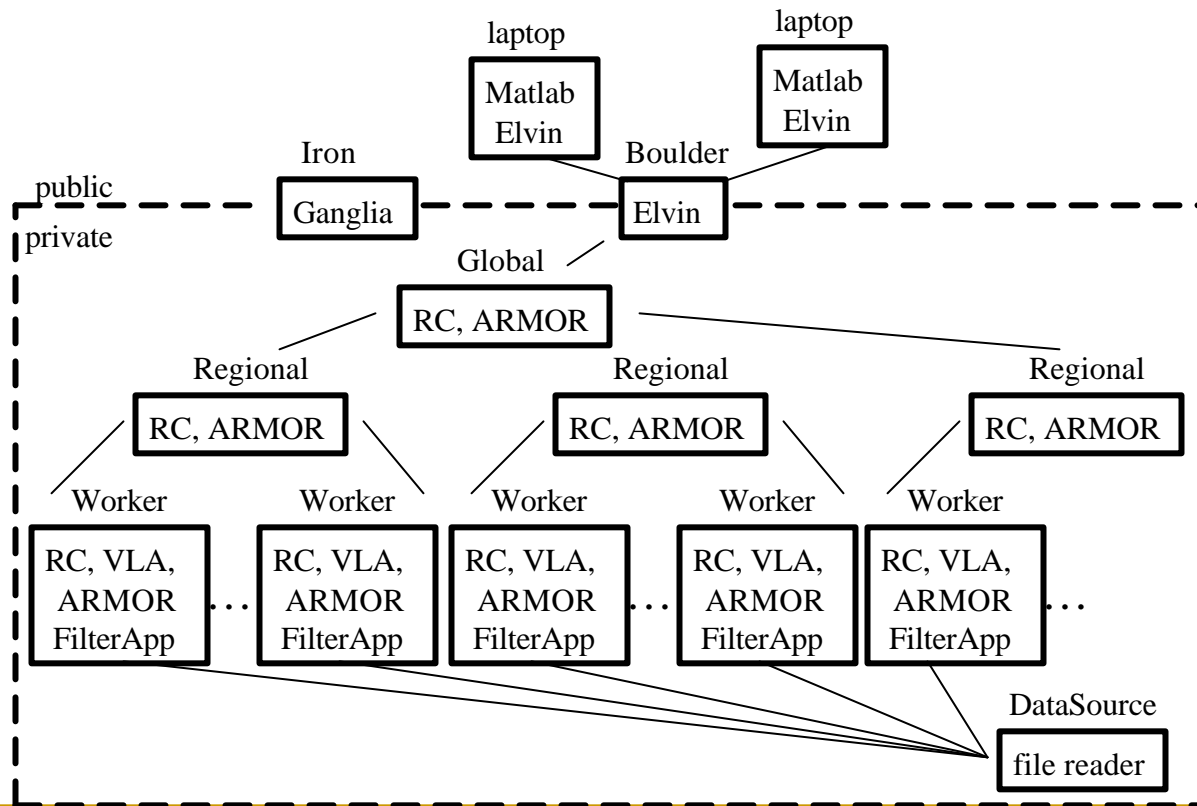
L2/3 Prototype Farm Components

- Using PC's from old PC Farms at Fermilab
 - 3 dual-CPU Manager PCs
 - boulder (1 GHz P3) - meant for data server
 - iron (2.8 GHz P4) - httpd, BB and Ganglia gmetad
 - slate (500 MHz P3) - httpd, BB
 - Managers have a private network through 1 Gbps link
 - bouldert, ironr, slatet
 - 15 dual-CPU (500 MHz P3) workers (btrigw2xx)
 - 84 dual-CPU (1GHz P3) PC workers (btrigw1xx)
 - No plans to add more, but may replace with faster ones
 - Ideal for RTES
 - 11 workers already have problems!

The Demonstration System Components

- Matlab as GUI engine
 - GUI defined by GME models
- Elvin publish/subscribe networking (everywhere)
 - Messages defined by GME models
- RunControl (RC) state machines
 - Defined by GME models
- ARMORs
 - Custom elements defined by GME models
- FilterApp, DataSource
 - Actual physics trigger code
 - File-reader supplies physics/simulation data to the FilterApp
 - Demo “faults” encoded onto Source-Worker data messages for execution on the Worker

The Demonstration System Architecture



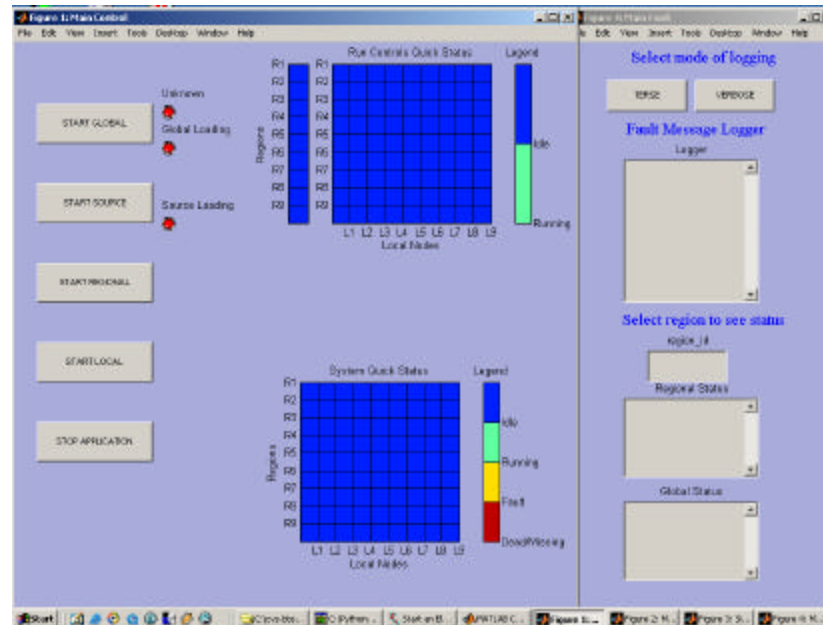
BB and Ganglia

- For traditional monitoring



<http://iron.fnal.gov/>

- For RTEs/Demo-specific monitoring



Comments

- This is an integrated approach – from hardware to physics applications
 - Standardization of resource monitoring, management, error reporting, and integration of recovery procedures can make operating the system more efficient and make it possible to comprehend and extend.
 - There are real-time constraints that must be met
 - Scheduling and deadlines
 - Numerous detection and recovery actions
 - The product of this research will
 - Automatically handle simple problems that occur frequently
 - Be as smart as the detection/recovery modules plugged into it
-

Comments (continued)

- The product can lead to better or increased
 - System uptime by compensating for problems or predicting them
 - instead of pausing or stopping the experiment
 - Resource utilization
 - the system will use resources that it needs
 - Understanding of the operating characteristics of the software
 - Ability to debug and diagnose difficult problems

Further Information

- General information about RTES
 - <http://www-btev.fnal.gov/public/hep/detector/rtes/>
- General information about BTeV
 - <http://www-btev.fnal.gov/>
- Information about GME and the Vanderbilt ISIS group
 - <http://www.isis.vanderbilt.edu/>

Further Information (continued)

- Information about ARMOR technology
 - <http://www.crhc.uiuc.edu/DEPEND/projects-ARMORs.htm>
- Talks from our last workshop
 - <http://false2002.vanderbilt.edu/program.php>
- Wiki (internal, today)
 - whcdf03.fnal.gov/BTeV-wiki/DemoSystem2004
- Elvin publish/subscribe networking
 - www.mantara.com

Demonstration Outline

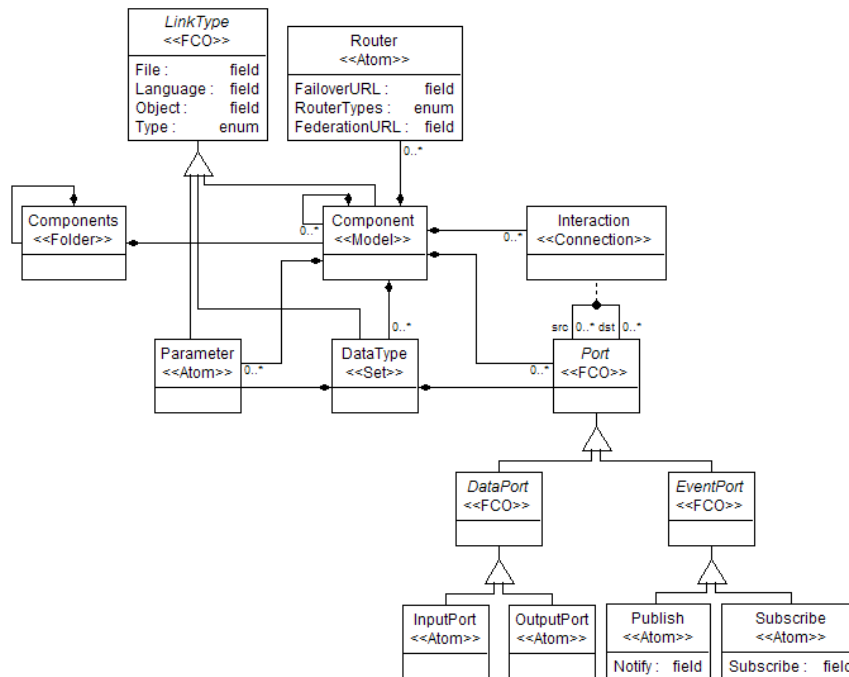
- GME
 - SIML, DTML, Custom Elements, Matlab GUI
 - Meta language
- Building and Deployment
- Runtime
 - ARMORs, VLAs
 - Fault detection and mitigation
- Reconfigure and Redeploy
 - 2x, 4x

Demonstration Slides

MIC Based System Modeling

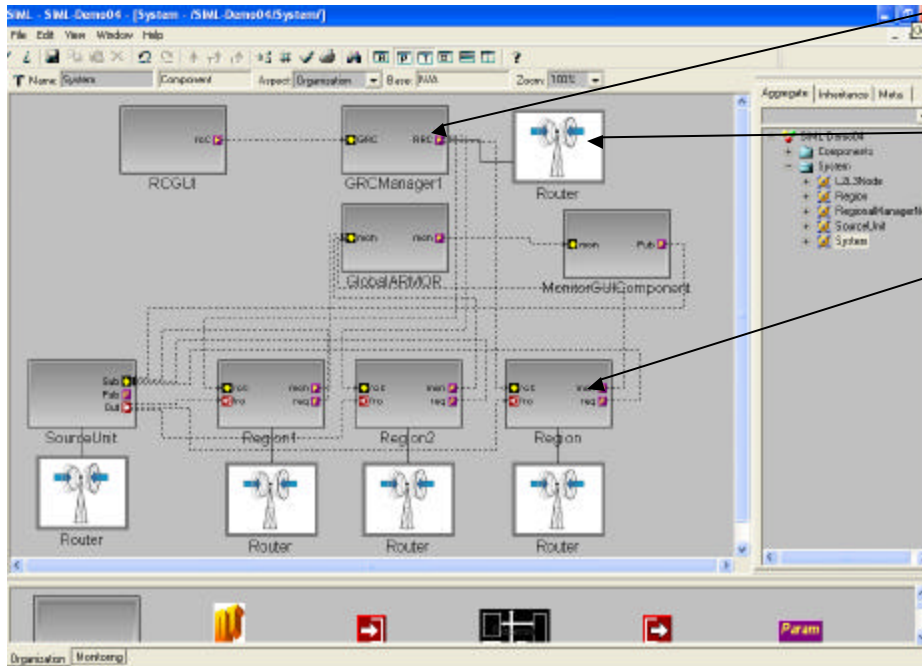
- Design a domain-specific modeling language
 - Provision concepts for all the different aspects of the system
 - Express their interactions
 - A “super” system-wide modeling language
- Implement a suite of translators
 - Generate fault-mitigation behaviors
 - Generate system build configurations
 - Link with user code/libraries

SIML –System Integration Modeling Language



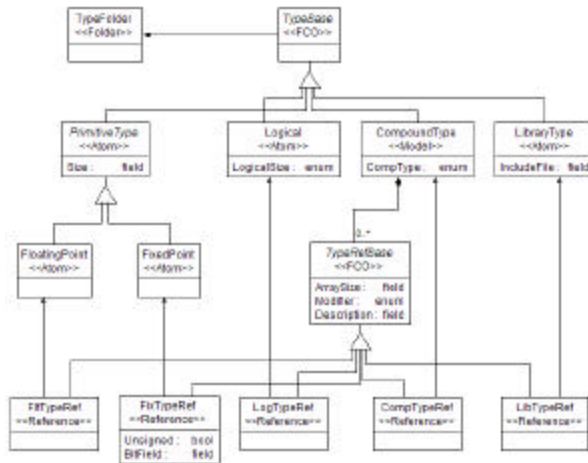
- Model Component Hierarchy and Interactions
- Loosely specified model of computation
- Model information relevant for system configuration

System Architecture

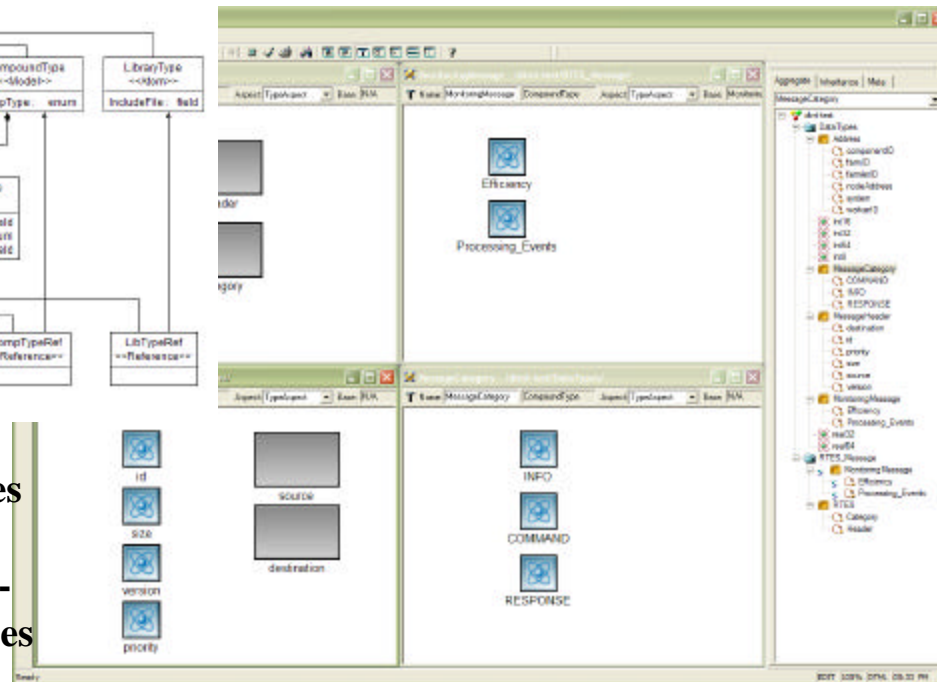


- RunControl Manager
- Router Information
- How many regions ?
- How many worker nodes inside the region?
- Node Identification information

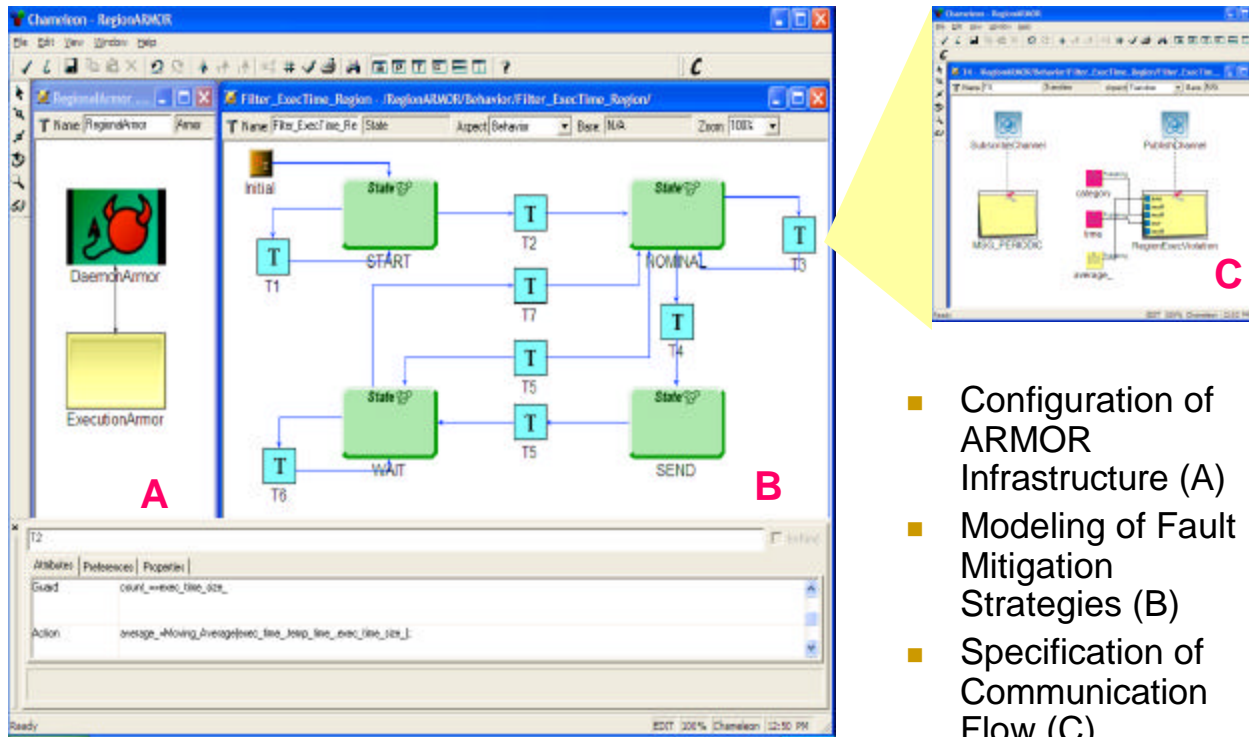
Data Type Modeling Language -DTML



- **Modeling of Data Types and Structures**
- **Configure marshalling-demarshalling interfaces for communication**

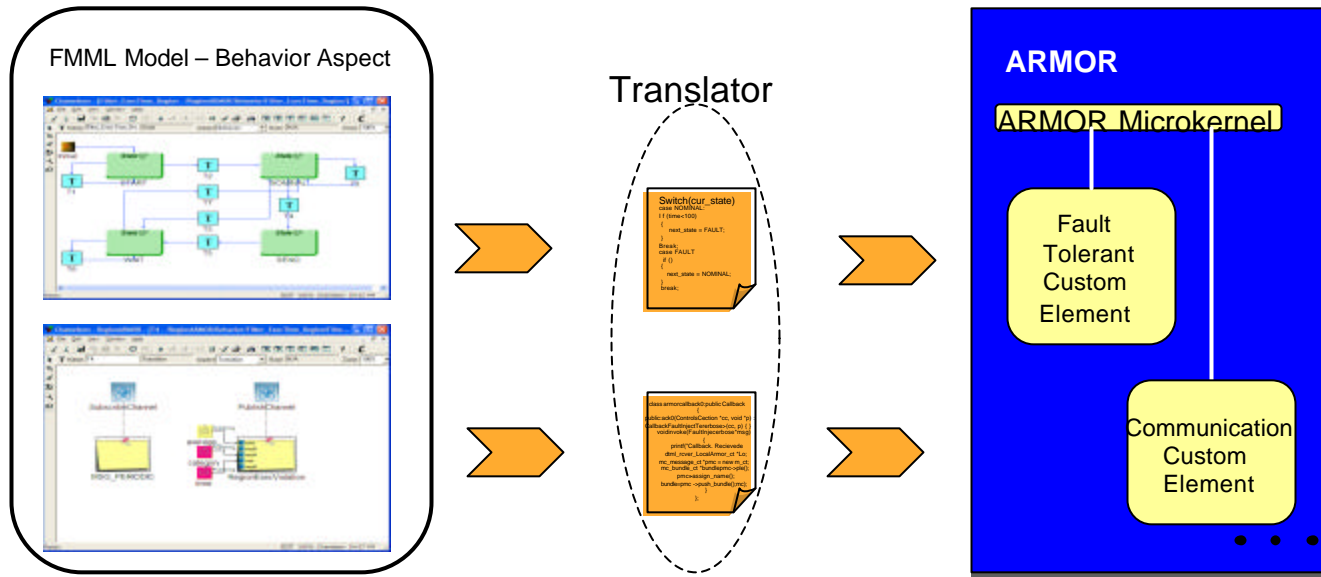


Fault Mitigation Modeling Language (1)



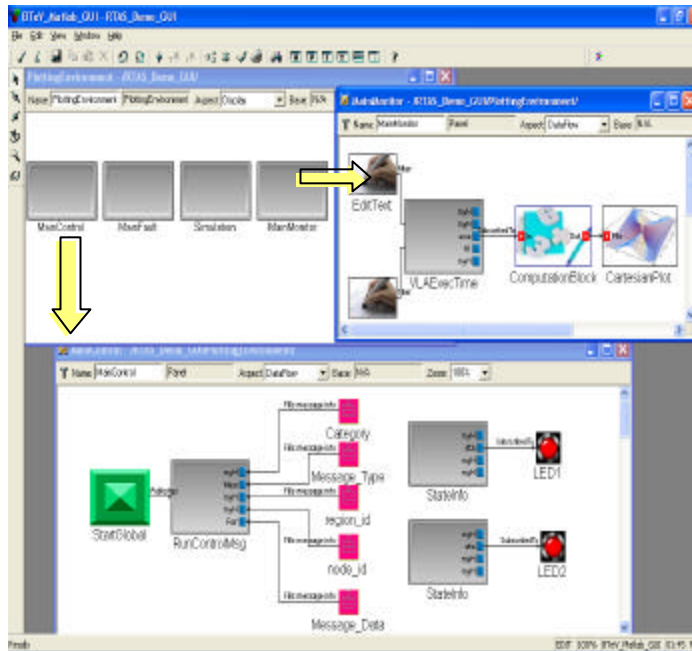
- Configuration of ARMOR Infrastructure (A)
- Modeling of Fault Mitigation Strategies (B)
- Specification of Communication Flow (C)

Fault Mitigation Modeling Language (2)



- Model translator generates fault-tolerant strategies and communication flow strategy from FMML models
- Strategies are plugged into ARMOR infrastructure as ARMOR elements
- ARMOR infrastructure uses these custom elements to provide customized fault-tolerant protection to the application

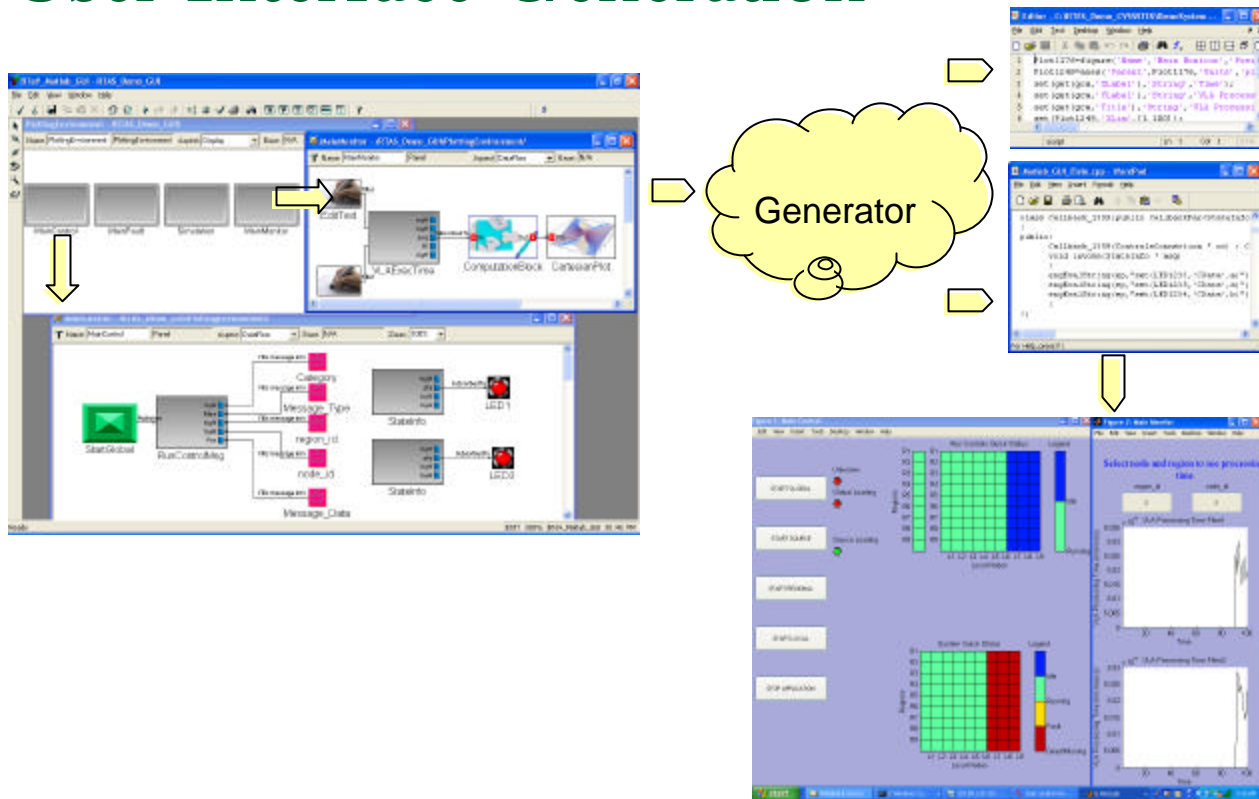
User Interface Modeling Language



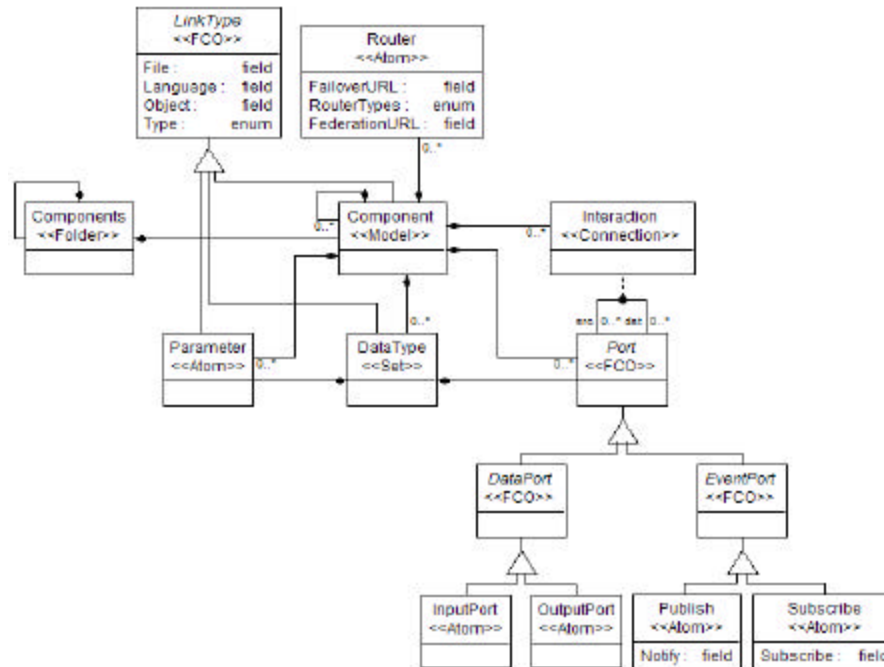
Example User Interface Model

- Enables reconfiguration of user interfaces
- Structural and data flow codes generated from models
- User Interface produced by running the generated code

User Interface Generation

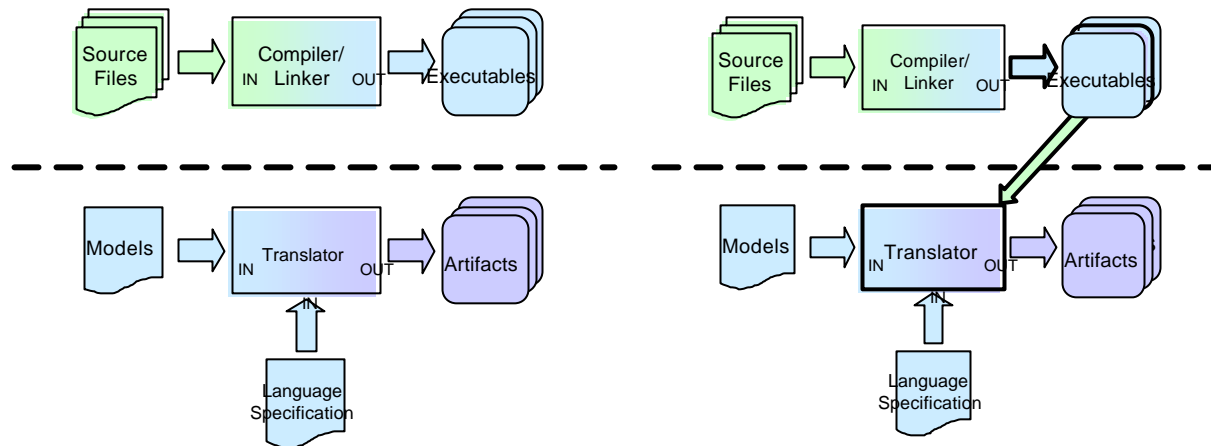


GME meta language(s) –Meta Model

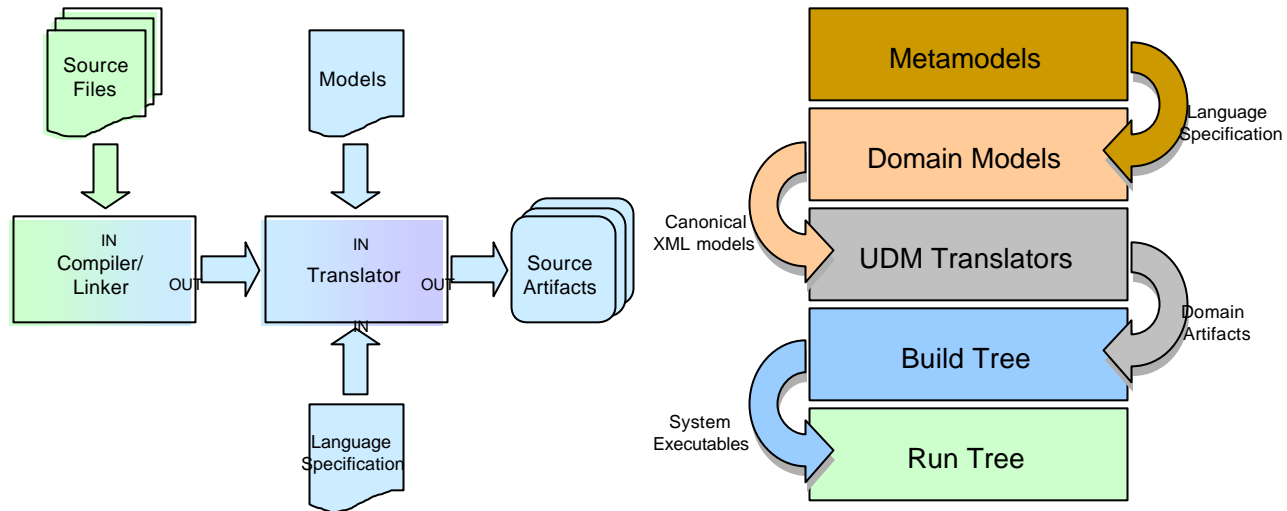


Versioning/Build System

- An equivalent XML representation of GME models is stored in the CVS tree
 - UDM tools provide forward and backward translation from MGA to XML
- Model transformers developed with UDM
 - Can be built for Windows and Linux platforms
 - Model transformation code is also stored in CVS



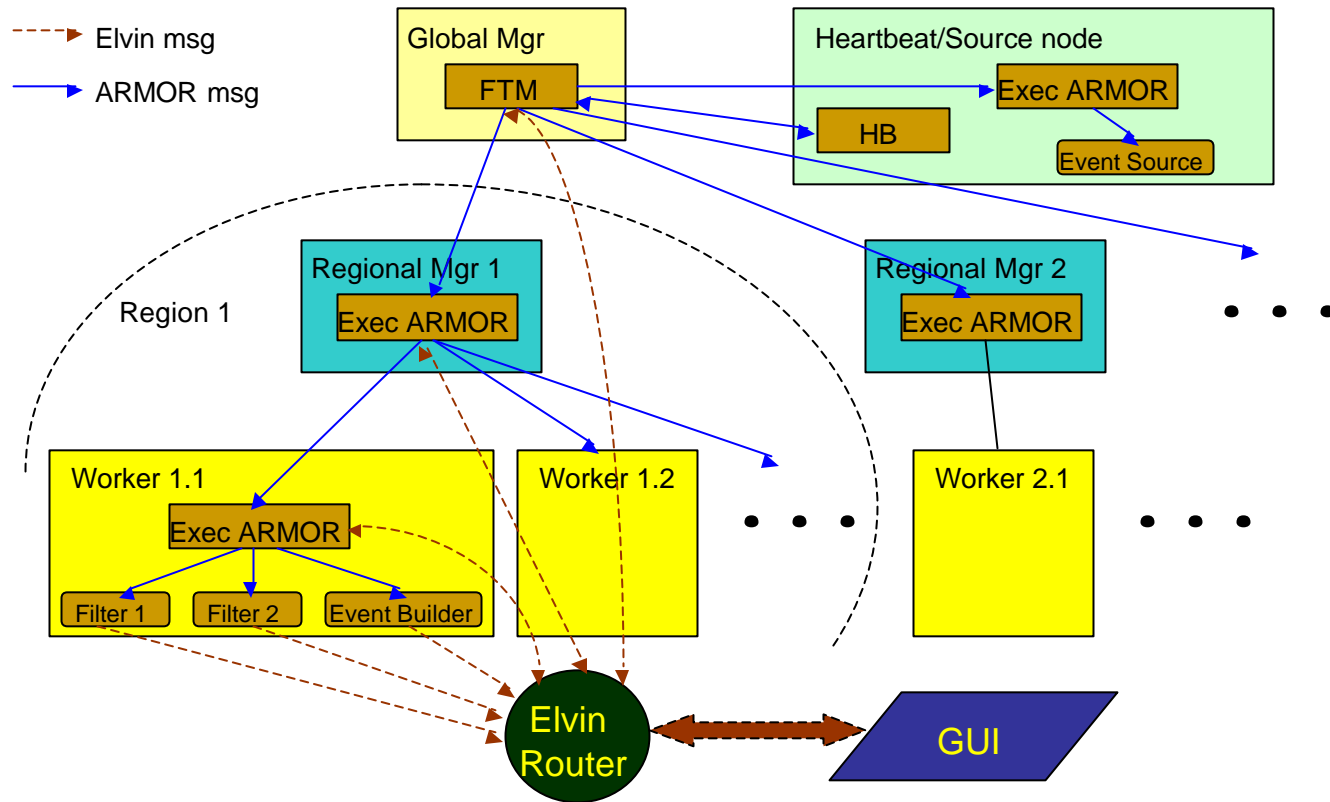
Versioning/Build System



Multi-stage build

1. Compiles model transformers
2. Makefiles invoke model transformers upon the stored models and generates code artifacts (behavior/config code)
3. Generated code place in the existing source tree
4. Source tree is compiled and linked to produce binaries
5. Additional tools take over to traverse the run tree and distribute the components to all the nodes

ARMOR Configuration in RTES



Execution ARMOR in Worker

